

To read the article online, visit <http://www.4GuysFromRolla.com/articles/091207-1.aspx>

Examining ASP.NET's Membership, Roles, and Profile - Part 9

By [Scott Mitchell](#)

Introduction

ASP.NET's Membership, Roles, and Profile systems were designed using the [provider model](#), which enables these systems to seamlessly use different implementations. ASP.NET ships with a provider for managing members and roles through SQL Server and another for using Active Directory. It is also possible to plug in other implementations that have been built from the ground up or downloaded from other sources. For example, you can download alternative providers from Microsoft that store membership and role information in a Microsoft Access database (see [Part 8](#) of this article series). Most of the articles in this series, however, have focused on using the SQL Server provider ([SqlMembershipProvider](#), [SqlRoleProvider](#), and [SqlProfileProvider](#)). The SQL Server providers are typically the provider of choice for Internet-based web applications, whereas the Active Directory providers are more commonly used in intranet scenarios.

The SQL Server providers create a number of tables, views, and stored procedures in the specified SQL Server database. Therefore when using these providers it is possible to add, modify, or delete membership or roles or profile-related data through T-SQL statements. In this article we'll look at a common membership need - deleting users. While users can certainly be deleted through the .NET Membership API, there are scenarios where it may be much easier to use a T-SQL script. However, bypassing the managed APIs and working directly with the database is not without its own host of challenges. By the end of this article we'll have addressed these issues, discussed the pros and cons of using T-SQL in lieu of the managed APIs, and have examined both the managed API methods and T-SQL commands for deleting a single user and deleting all users. Read on to learn more!

Deciding Whether to Work with the Programmatic APIs or Whether to Work with SQL Server Directly

The SQL Server providers create the tables, views, and stored procedures needed by the Membership, Roles, and Profile systems in a specified SQL Server database. These database objects are created automatically in the `ASPNETDB.MDF` database in the `~/App_Data` folder when using the [ASP.NET Website Administration Tool](#). Alternatively, these objects can be explicitly added via the [ASP.NET SQL Server Registration Tool](#) ([aspnet_regsql.exe](#)) (see [Part 3](#)).

.NET's Membership, Roles, and Profile APIs offer a host of methods for programmatically creating, modifying, and deleting user accounts. However, when using the SQL Server providers it may be tempting to modify, add, or delete user accounts directly through T-SQL commands rather than through code. These API methods call the applicable stored procedures. Bypassing the APIs and stored procedures and modifying table data manually has the potential to lead to incorrect or corrupted data. For example, in the Membership table each user's password is stored. If the Membership system is configured to use a hash of the password, modifying the password incorrectly (or modifying its salt) can essentially lock out that user since their plain-text password no longer matches up to the stored password hash. In short, if you decide to work with the user information directly from T-SQL commands, it is best to do so through the stored procedures explicitly added by the provider. Moreover, it is essential to have a solid understanding of the data model and the stored procedure(s) you invoke.

There are several advantages to using the managed APIs as opposed to making modifications to the

database directly. For one, the managed APIs are much easier to understand than the SQL Server-based providers' stored procedures and data model. The methods and properties in the Membership, Roles, and Profile APIs are well-documented on Microsoft's website and the property names and method names and input parameters can often be understood just by their names and through the short IntelliSense description. The SQL Server stored procedures, on the other hand, are far less documented and require parsing the contents to fully understand what's happening.

The downside of the managed APIs is that you must write code to perform the action and this code must somehow be executed. Typically, such code will be placed in a web page in some Administration section on a site, requiring a user to logon to the site, visit the Administration section, and execute the desired command (such as deleting a particular user or removing all users from the database). The upside of issuing T-SQL commands is that they can more easily be invoked. Imagine that you have a test server with some artificially created accounts. When rolling out a new version of the software you may want to remove all account data and recreate new test accounts. This can be accomplished by executing a T-SQL script with the appropriate commands to remove the current users and add in the test users.

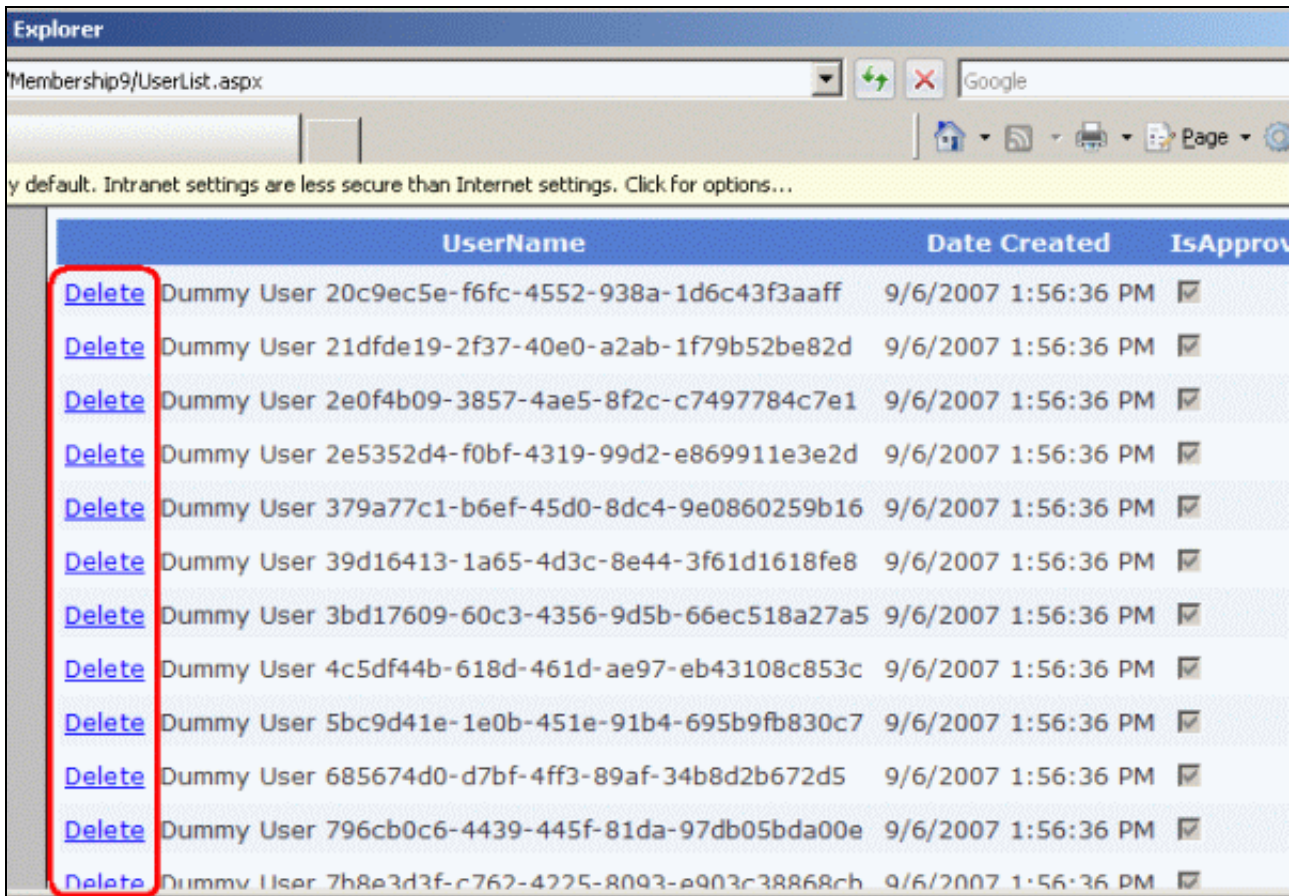
The remainder of this article looks at how to delete a single user and delete all users using both the managed APIs and T-SQL commands.

Deleting a Single User

To delete a single user using the managed APIs, use the [Membership.DeleteUser\(username\[, deleteAllData\] method](#). This method takes, at minimum, a single string input - the username of the user to delete. It can also accept an optional Boolean parameter that indicates whether or not to delete all related user information, such as their role information, profile data, and so on. If you do not explicitly provide this Boolean parameter, a value of True is used. For the SQL Server provider, the `Membership.DeleteUser` method invokes the stored procedure `aspnet_Users_DeleteUser`, which expects the following parameters:

- `@ApplicationName` - the application which the specified user is found and is to be deleted from. As discussed in [Part 1](#) of this article series, the membership system is partitioned into applications by application names. This allows for multiple web applications to use a common member store.
- `@UserName` - the name of the user to delete.
- `@TablesToDeleteFrom` - an integer field that specifies what membership-related tables to delete data from. This parameter serves as a bit field that indicates what subset of tables to remove membership data from. If you pass in a value of 1, data is only deleted from the `aspnet_Membership` table; the user is *not* removed from the `aspnet_Users` table. Not surprisingly, if you call `Membership.DeleteUser(username, False)`, a value of 1 is passed to this parameter. Other values for this parameter can specify that the stored procedure should delete the record from other combinations of tables. Passing in a value of 15 (which is what is passed in when calling `Membership.DeleteUser(username)` or `Membership.DeleteUser(username, True)`), deletes all user related records from the user account-related tables, including `aspnet_Users`. In short, passing in a value of 15 deletes the user entirely from the database.
- `@NumTablesDeletedFrom` - an output parameter that returns the number of tables where data was deleted from.

The demo available for download at the end of this article includes a web page that lets the visitor view all of the users in a drop-down list. They can pick a user and then opt to delete him by clicking a button. Doing so deletes the user by calling `Membership.DeleteUser(username)`.



Membership9/UserList.aspx

by default. Intranet settings are less secure than Internet settings. Click for options...

	UserName	Date Created	IsApproved
Delete	Dummy User 20c9ec5e-f6fc-4552-938a-1d6c43f3aaff	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 21dfde19-2f37-40e0-a2ab-1f79b52be82d	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 2e0f4b09-3857-4ae5-8f2c-c7497784c7e1	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 2e5352d4-f0bf-4319-99d2-e869911e3e2d	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 379a77c1-b6ef-45d0-8dc4-9e0860259b16	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 39d16413-1a65-4d3c-8e44-3f61d1618fe8	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 3bd17609-60c3-4356-9d5b-66ec518a27a5	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 4c5df44b-618d-461d-ae97-eb43108c853c	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 5bc9d41e-1e0b-451e-91b4-695b9fb830c7	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 685674d0-d7bf-4ff3-89af-34b8d2b672d5	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 796cb0c6-4439-445f-81da-97db05bda00e	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>
Delete	Dummy User 7h8e3d3f-c762-4225-8093-e903c38868cb	9/6/2007 1:56:36 PM	<input checked="" type="checkbox"/>

To delete a particular user through T-SQL, simply call the `aspnet_Users_DeleteUser` stored procedure passing in the appropriate parameters.

```
DECLARE @NumTablesDeletedFrom int
EXEC aspnet_Users_DeleteUser applicationName, userName, 15, @NumTablesDeletedFrom OUTPUT

PRINT 'Number of tables deleted from: ' + @NumTablesDeletedFrom
```

To determine the application's name, look in the `aspnet_Applications` table. As discussed in [Part 1](#) you should be explicitly specifying the application name via `Web.config`. See [Always Set the "applicationName" Property When Configuring ASP.NET 2.0 Membership and Other Providers](#) article for further details and explanation.

Deleting All Users

The Membership API does **not** include a `Membership.DeleteAllUsers()` method. Nor is there a stored procedure in the SQL Server provider that deletes all users. But with a little elbow grease we can implement this functionality ourselves. The Membership API does provide a method to get all users in the system - `Membership.GetAllUsers`. We can use this method to get a list of users back and then loop through the users one at a time calling the `Membership.DeleteUser(username)` method.

```
' Visual Basic
'Enumerate each user and delete him!
For Each usr As MembershipUser In Membership.GetAllUsers()
    Membership.DeleteUser(usr.UserName)
Next

// C#
foreach (MembershipUser usr in Membership.GetAllUsers())
```

```
Membership.DeleteUser(usr.UserName);
```

A similar approach can be used to delete all users directly through T-SQL commands. You can create a CURSOR that enumerates the set of users found in the `aspnet_Users` table and then for each record call the `aspnet_Users_DeleteUser` stored procedure. For more on CURSORS see [Using SQL Server Cursors](#).

There's an easier approach, however: simply delete all of the records from all of the corresponding tables. The only challenge here is that there are foreign key constraints between related tables so you have to be certain to delete the data from the "child" tables before deleting the data from the "parent" tables. The following string of `DELETE` commands deletes the data in the correct order:

```
DELETE FROM dbo.aspnet_Membership
DELETE FROM dbo.aspnet_UsersInRoles
DELETE FROM dbo.aspnet_Profile
DELETE FROM dbo.aspnet_PersonalizationPerUser
DELETE FROM aspnet_Users
```

As a final option, you could remove all users by obliterating the member store by reinstalling the SQL Server provider database objects via `aspnet_regsql.exe`.

Happy Programming!

- By [Scott Mitchell](#)

Attachments

- [Download the code used in this article](#)

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 9
Article Author:	Scott Mitchell
Published Date:	September 12, 2007
Article URL:	http://www.4GuysFromRolla.com/articles/091207-1.aspx

Copyright 2014 QuinStreet Inc. All Rights Reserved.
[Legal Notices](#), [Licensing](#), [Permissions](#), [Privacy Policy](#).
[Advertise](#) | [Newsletters](#) | [E-mail Offers](#)